**PLAN DE TRABAJO – Rol: Infraestructura & MLOps**

**Responsable:** Edwar Marín  
**Duración:** 10–12 días  
**Objetivo:** asegurar que el proyecto sea **ejecutable, medible, reproducible y seguro**, con pipelines automatizados y métricas claras.

**🔹 FASE 1 — Preparar entorno reproducible (Días 1–3)**

**🎯 Objetivo:**

Garantizar que cualquiera pueda ejecutar el proyecto con un solo comando (docker compose up).

**✅ Tareas:**

1. **Crear requirements.txt**
   * Exportar dependencias actuales:
   * pip freeze > requirements.txt
   * Asegurarte de incluir: fastapi, uvicorn, streamlit, requests, pydantic, y más adelante langchain, chromadb, etc.
2. **Crear Dockerfiles**
   * Uno para el **backend** (backend/Dockerfile)
   * Uno para el **frontend** (Dockerfile o ui/Dockerfile)
   * Ejemplo para backend:
   * FROM python:3.11
   * WORKDIR /app
   * COPY ./backend /app
   * COPY requirements.txt .
   * RUN pip install -r requirements.txt
   * EXPOSE 8000
   * CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
3. **Configurar docker-compose.yml**
   * Define servicios:
     + api: para FastAPI
     + ui: para Streamlit
     + db-vector: para Chroma o FAISS (más adelante)
   * Ejemplo base:
   * version: "3"
   * services:
   * api:
   * build: ./backend
   * ports:
   * - "8000:8000"
   * volumes:
   * - ./backend:/app
   * env\_file:
   * - .env
   * ui:
   * build: .
   * ports:
   * - "8501:8501"
   * depends\_on:
   * - api
4. **Verificación**
   * Ejecutar:
   * docker compose up --build
   * Confirmar que ambos servicios (API y UI) funcionan desde el navegador.

**🔹 FASE 2 — Monitoreo y logs anonimizados (Días 4–6)**

**🎯 Objetivo:**

Medir rendimiento sin almacenar información sensible.

**✅ Tareas:**

1. **Crear módulo utils/logger.py** para registrar:
   * session\_id aleatorio (UUID)
   * timestamp
   * question\_length, latency, tokens, model\_name, cost\_estimate
   * Guardar en CSV o JSON (logs/metrics.csv)
2. import time, uuid, json
3. from datetime import datetime
4. def log\_interaction(question, model, latency, cost):
5. entry = {
6. "session\_id": str(uuid.uuid4()),
7. "timestamp": datetime.now().isoformat(),
8. "question\_length": len(question),
9. "model": model,
10. "latency\_ms": latency,
11. "cost\_usd": cost
12. }
13. with open("logs/metrics.jsonl", "a") as f:
14. f.write(json.dumps(entry) + "\n")
15. **Integrar logs en main.py**
    * Al final del endpoint /query, antes del return, llama a log\_interaction().
    * Usa time.time() para medir latencia.
16. **Anonimización**
    * No almacenar la pregunta ni IP.
    * Solo datos estadísticos.
17. **Visualización simple (opcional)**
    * Crear un script analyze\_logs.py que lea metrics.jsonl y muestre promedio de latencia, número de consultas, etc.

**🔹 FASE 3 — Seguridad y entorno controlado (Días 7–8)**

**🎯 Objetivo:**

Asegurar manejo correcto de llaves y evitar exposición de datos.

**✅ Tareas:**

1. **Crear .env y .env.example**
2. OPENAI\_API\_KEY=your\_api\_key\_here
3. DB\_PATH=data/vectorstore
4. DEBUG=False
5. **Usar python-dotenv** para cargar variables:
6. pip install python-dotenv

Y en main.py:

from dotenv import load\_dotenv

load\_dotenv()

1. **Agregar .env al .gitignore**:
2. .env
3. logs/
4. \_\_pycache\_\_/
5. **HTTPS (opcional si hay tiempo):**
   * Simular HTTPS local usando un proxy inverso (Nginx) o usar Cloudflare Tunnel para demo.

**🔹 FASE 4 — Métricas automáticas y evaluación (Días 9–10)**

**🎯 Objetivo:**

Medir automáticamente la calidad de respuestas del ChatBot con el conjunto de preguntas *gold*.

**✅ Tareas:**

1. **Crear script evaluate.py**
   * Leer data/gold\_questions.csv
   * Llamar al endpoint /query para cada pregunta
   * Guardar resultados (exactitud, cobertura, citas válidas, alucinación, etc.) en CSV
2. import csv, requests, time
3. API\_URL = "http://localhost:8000/query"
4. with open("data/gold\_questions.csv", newline="", encoding="utf-8") as f:
5. reader = csv.DictReader(f)
6. results = []
7. for row in reader:
8. q = row["question"]
9. start = time.time()
10. r = requests.post(API\_URL, json={"question": q, "mode": "breve"})
11. latency = time.time() - start
12. data = r.json()
13. results.append({
14. "question": q,
15. "latency": latency,
16. "answer": data.get("answer", ""),
17. "citations": len(data.get("citations", []))
18. })
19. with open("logs/evaluation\_results.csv", "w", newline="", encoding="utf-8") as f:
20. writer = csv.DictWriter(f, fieldnames=results[0].keys())
21. writer.writeheader()
22. writer.writerows(results)
23. **Pipeline automático (si quieres usar n8n o GitHub Actions)**
    * Crear flujo que corra el evaluate.py periódicamente o tras cada commit.
    * Exportar métricas en CSV o dashboard simple.

**🔹 FASE 5 — Documentación técnica (Día 11–12)**

**🎯 Objetivo:**

Entregar evidencia técnica reproducible.

**✅ Tareas:**

1. Documentar en el README.md:
   * Cómo levantar el sistema (docker compose up).
   * Estructura de carpetas.
   * Ejemplo de logs y métricas.
   * Variables de entorno (.env.example).
   * Cómo ejecutar el script de evaluación.
2. Crear un **reporte técnico breve** (2–3 páginas) describiendo:
   * Qué cubre tu rol (infra, métricas, seguridad).
   * Cómo garantizas reproducibilidad y trazabilidad.
   * Ejemplo de resultados (latencia promedio, costo, etc.).

**🧩 Resultado esperado al final del plan**

| **Componente** | **Estado esperado** |
| --- | --- |
| Docker y compose | ✅ Reproducible |
| Logs anonimizados | ✅ Implementados |
| Métricas básicas | ✅ CSV o JSON |
| Evaluación automática | ✅ Script evaluate.py |
| Seguridad (.env) | ✅ Activa |
| Documentación | ✅ Lista para entrega |